The University of Alberta
Printing Department
Edmonton, Alberta

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR Robert Gordon Donaldson

TITLE OF THESIS Implications of Microprogramming.

DEGREE FOR WHICH THESIS WAS PRESENTED M.Sc.

YEAR THIS DEGREE GRANTED 1973

THE UNIVERSITY OF ALBERTA

IMPLICATIONS OF MICROPROGRAMMING

BY

ROBERT GORDON DONALDSON

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1973

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled IMPLICATIONS OF MICROPROGRAMMING submitted by Robert Gordon Donaldson in partial fulfillment of the requirements for the degree of Master of Science.

ABSTRACT

The purpose of this thesis is to examine current microprogramming usage and consider future application areas. A survey of past usage and an explanation of a typical microprcgramming approach is presented. Comparisons of microprogramming with software and hardware are abstracted for their relative advantages and disadvantages. Present applications are reviewed for user and manufacturer benefit. Future usage is extrapclated from present usage and current predictions for the industry in both the hardware and software segments.

ACKNOWLEDGMENTS

I would like to acknowledge the assistance and guidance of Dr. T. A. Marsland in the preparation of this thesis. The financial support received from the Department of Computing Science is gratefully appreciated. I would also like to thank my wife Sherry, whose patience and typing have made this possible.

TABLE OF CONTENTS

Page

CHAPTER I. TERMINOLOGY 1

1.1 Introduction 1

1.2 General Considerations and Definitions 1

1.3 Summary 6

CHAPTER II. HARDWARE - SOFTWARE CONSIDERATIONS 7

2.1 Introduction 7

2.2 Historical Review 7

2.3 Microprogramming vs Hardware 17

2.4 Microprogramming vs Software 20

2.5 Summary 22

CHAPTER III. APPLICATIONS 23

3.1 Introduction 23

3.2 Manufacturers Applications 24

3.3 User Applications 30

3.4 Computer Design 34

3.5 Summary 35

Page

CHAPTER IV. FUTURE DEVELOPMENTS 40

4.1 Introduction 40

4.2 Power vs Flexibility 41

4.3 Developing Architectures 42

4.4 Current Areas 46

4.5 System Dependent Features 50

4.6 Summary 53

CHAPTER V. CONCLUSIONS 54

REFERENCES 61

LIST OF ILLUSTRATIONS

Figure Page

2.1 Wilkes Original Microprogram Control 9

2.2 MLP-900 General Configuration 14

2.3 MLP-900 Inner Computer Configuration 14

CHAPTER I

TERMINOLOGY

1.1 Introduction

Microprogramming has developed as a key field in Computing Science over the past decade. How, why, and when this development came about is the concern of the first part of this thesis. The role that it plays in the eyes of the manufacturer and user is considered in Chapter III. Future areas of application in current systems, language processors, system performance monitors, telecommunications equipment, minicomputers and time sharing systems are evaluated in terms of microprogram control. Developing architectures are considered for their use of microprogramming. Chapter V restates the disadvantages of microprogramming and examines the various user categories for applicability.

1.2 General Considerations and Definitions

The concept of microprogramming was first introduced by Wilkes [35] in 1951. In his paper The Best Way to Design an

Automatic Calculating Machine he proposed a systematic procedure for computer control-logic design. His approach considered the execution of an instruction as a series of sequential and/or parallel register to register transfers which he likened to the execution of steps in a program (including state dependent transfers). The steps of a microprogram are microinstructions.

Wilkes' original idea was to use microprogramming to aid in the design of computer control-logic with a read only memory (ROM) technique. He noted the possibility of a read/write control memory but questioned the need for such a design. In a later literature survey on microprogramming, Wilkes [36] recognizes the validity of a read/write memory.

A microprogram's constituent instructions are fetched from a read only control memory whose read time approximates the CPU cycle time. ROM cannot be changed dynamically as opposed to the read/write memories mentioned, which provide for dynamically alterable instruction sets, and user and manufacturer microroutine libraries. The application of a dynamic writable control store introduces similar problems to those encountered with operating systems and main memory management. Problems that will have to be approached are, the development of paging techniques for writable control store management, interrupt handlers for executing microprograms and dispatching algorithms for these same

microprograms.

A formal definition of microprogramming is provided by Husscn in his book Microprogramming: Principles and Practices as:-

> ... a technique fcr designing and implementing the control function of a data processing system as a sequence of control signals, to interpret fixed or dynamically changeable data processing functions. These control signals, organized on a word basis and stored in a fixed or dynamically changeable control memory, represent the states of the signals which contrcl the flow of information between the executing functions and the orderly transitiicn between these signal states.

The definition is consistent with Wilkes' concept of microprogramming but goes further to include dynamically alterable ccntrol memory, it is the definition used throughout the remainder cf the thesis.

Distinct from the control function of a computer system is the system architecture which Husson [12] defines as,

> ... attributes of the system as seen by the programmer. It represents the conceptual structure of the functional behaviour of the system as distinct from the organizatiicn of the data flcw controls, logical design, and physical implementation.

While the architecture is distinct from the control function of the computer, it is defined by the microprograms that perform the ccntrol function. The term host machine is used to represent the hardware machine while virtual machine is the architecture defined cn the host machine.

Microprogramming is a tool for both design and implementation, that can be used to great advantage in realizing a machine's (or a group of machines') architecture. In addition to the realization of a particular system, microprogramming can allow a system to emulate the architecture of another machine. Rosin [26] defines an emulator as;

> ... a complete set of microprograms which, when imbedded in a control store, define a machine.

The definition includes the basic set of microprograms as an emulator for the machine.

In addition to the word microprogramming, other terms have also been used to express that concept. Opler [17] used the term 'firmware' to represent microprograms which are used beyond the original architectural designs. The term 'easyware' has been used, but maybe even more apt is 'underware'.

Redfield in his paper, A Study in Microprogrammed Processors: A Medium Sized Microprogrammed Processor [21], sets out several characterizing definitions which give a means of describing microprogammed CPUs with consistent terminology:-

(a) The terms monophase and polyphase describe the duration of a microinstruction's control of the CPU data

paths. A monophase instruction controls the execution for one clock cycle whereas polyphase instructions have a duration of two or more clock pulses. These two techniques have sometimes been referred to as vertical and horizontal microprogramming. Generally they are considered distinct strategies for the optimization of the control memory. Vertical microprogramming optimizes the microinstruction set available in the CPU, while horizontal microprogramming provides for the optimal generation of control signals within the CPU (sometimes referred to as soft and hard microprogramming respectively).

(b) Parallel and serial execution describe the overlapping of microinstruction execution. Under parallel execution, some form of instruction look-ahead is used at branch points, with the addressing of the next microinstruction overlapping the execution of the present instruction. In serial execution, the addressing of the next instruction is done only when the data path manipulation of the current instruction is finished.

(c) Encoding and little encoding describe the format of the microinstruction. With encoded control, a minimum number of bits is used to represent uniquely the required instuction set. Little encoding can range from no encoding to the encoding of mutually exclusive signals into one field (field encoding). The range of encoding is continuous from

none(1 bit in the microinstruction per data path) to complete encoding(log2 N+1 bits for N data paths).

The concepts of a microprogram, microinstructions, emulation, computer architecture, virtual and host machines are fundamental to a study of microprogramming. The terms monophase, polyphase, serial, parallel and encoding provide a terminology for comparison of microprogrammed machines.

1.3 Summary

Microprogramming in present and developing systems has large application, for reasons which will be looked into later. The future of computer systems in general will ultimately decide the application areas of microprogramming. Technological developments in the fields of memories, peripheral processors and telecommunications equipment will dictate an increased usage in computer systems or a reduction in application. The position of microprogramming is one of flux, it will either grow or be replaced but not remain at a constant level of application. The assessment of the future position of microprogramming relative to the computer industry in general is the aim of this thesis.

CHAPTER II

HARDWARE - SOFTWARE CONSIDERATIONS

2.1 Introduction

This chapter will give a historical review of microprogramming to orient the reader to the past developments in the field. The state of the art will be presented in the form of a brief literature review followed by a comparison of microprogramming to hardware and software. This comparison will illustrate the advantages and disadvantages relative to the traditional components of computer systems.

2.2 Historical Review

Historical reviews of computer science are available to the reader in more complete form than this author will try to assemble. Some of the better reviews being, Rosen's Electronic Computers: A Historical Survey, Rosen's Programming Systems and Languages: A Historical Survey, and Wilkes' The Growth Of Interest In Microprogramming: A Literature Survey. In addition to general historical

surveys, the reader has many papers of narrower scope available in most areas of computer science. These papers provide excellent bibliographies pointing to the background work in a specific area of interest. The approach taken here is to provide the reader with the essentials, so that he may see the reasons for microprogramming's position in computer systems today.

It was pointed out in Chapter I that Wilkes provided the idea for control-logic design. His first suggested design is worth examining for historical reasons, as well as the clarity with which it presents the concept. Figure 2.1 gives the structure of his microprogram control. The control-logic contains a register R(n), a decoding tree, clock signal, control matrix C, a sequencing matrix S, and decision logic (conditional flip-flop).

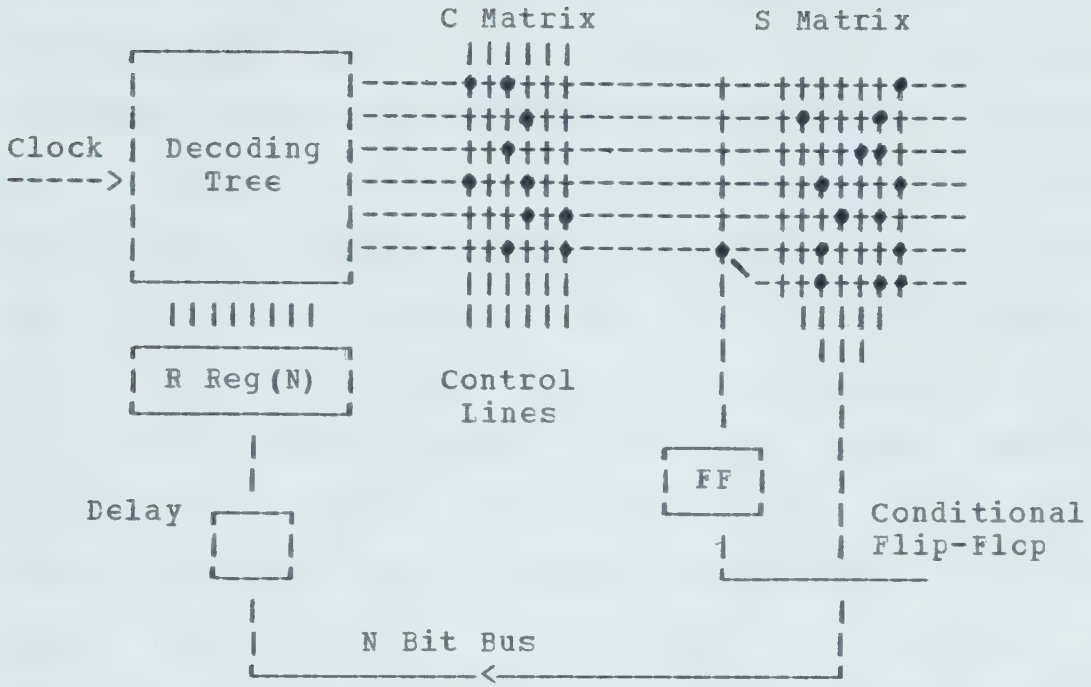

Figure 2.1: Wilkes Original Microprogram Control

Register R is initially loaded from an external source with the address in control memory of the starting microinstruction for the next microroutine. On a clock pulse register R is gated into the decoding tree to select the next unique microinstruction (activates a single horizontal line). The horizontal line feeds into the C matrix of vertical control lines which may be interconnected

by impedence couplers causing the appropriate control lines to be activated. These control lines are connected to the CPU's physical units (such as adders, shifters and control registers), either through a decoding sequence or directly. The S matrix is pulsed by the same microinstruction generating the address of the instruction to be executed next. This selection can be made conditional by the use of logic to combine an output from one or many elements in the CPU, via a flip-flop(FF), with the active horizontal line(IBM 360/50 supports 64 way branching). The address is delayed back to the R register for execution in the next cycle. Control memory is organized in sequences of microinstructions which perform the desired functions (e.g. diagnostic tests, multiply, divide or special application dependent routines). In this scheme no decoding is done to pulse the control lines, it is a direct one to one translation. In Redfield's [21] notation this would be considered, monophase, serial and little encode. The width of the ROM word is given by the number of lines emanating from the C and S matrices. The width of the S matrix is N the same as the R register. The C matrix width is dependent on the amount of parallelism, encoding and whether a polyphase technique is employed in the CPU.

Husson [12] in Chapter II gives an evolutionary description of the modifications and expansions made to

Wilkes' original idea. He works through a series of diagrams explaining the effects of modifications and the reasons for them.

Microprogramming was first implemented by Wilkes and his group at the University of Cambridge on the EDSAC 2. Several other one shot implementations were done within the next few years including the CIRRUS [1] computer, but the first commercial use was by IBM on their 7950, a Stretch computer with special high speed file processing equipment [36]. The stored logic was used to complement the conventional instruction set. Users could define complex file processing sequences (in the stored logic) and use them as instructions. With the announcement of their system 360 series, in 1964 IBM introduced the largest example of stored logic control. Tucker [32] gives a complete description of system 360 microprogramming and several reasons for its use:

1. Emulation - microprogrammed simulators for old systems on the newer system 360. Typically IBM provided emulators for the 1400 series equipment on the 360/20 and 7094 emulators on the 360/65 (additional hardware is used to facilitate special 7094 operations in the emulator).
2. Compatibility - uniform instruction sets from the smallest computer in a series to the

largest. The smaller CPUs using more microprogramming to implement the same features that are hardware on the larger systems in the series.

3. Flexibility - changes in the future can be effected with less expense than required to replace and redesign hardware control-logic. IBM had reason to be pleased with their decision when they decided to change their real arithmetic routines. This change was made quickly and at minimal expense to IBM [26].

4. Microdiagnostics - hardware diagnostic microroutines were included. These routines have greater resolving power than software diagnostics in the detection of errors [19].

In 1965 RCA announced the Spectra 70 series of computers of which the model 45 was microprogrammed. Benjamin [4] explains the approach taken by RCA in microprogramming the Spectra 70 model 45. The 45 was microprogrammed to be architecturally similar to the IBM 360. Control memory was organized with two stacks, 1024 words(56 bits), with interleaving between stacks. The access time of one stack being 960ns but with interleaving the effective access time is reduced to 480ns. Husson [12] goes into the detail of the structure and microprogramming technique used.

Several new computer systems were introduced following system 360 with quite extensive microprogramming facilities. In 1967, Interdata introduced the model 3 which had a transformer read only control store. Interdata then in quick succession introduced two other models, 4 (1968) and 5 (1970), both having a similiar type of control memory to the model 3. Subsequently several other manufacturers introduced microprogrammed models, Cincinnati Milacron CIP-2000 (ROM), Micro Systems Micro 800 (ROM), and Spiras model 65 (ROM) (information taken from Clapp [5]).

The first major system to be introduced with a separate writable control store (WCS) was Standard Computer Corporation's IC-6000E in 1967. In 1970 Standard Computer Corporation introduced the most dramatic example of this type of control, the MLP-900 using what Rakoczi [20] coined as an inner computer. Figure 2.2 shows a typical configuration with the inner computer as a separate unit within the system. In the inner computer, shown in Figure 2.3, more than the conventional control-logic is present. The inner computer's structure includes a scheduler, translators and decoders. The scheduler controls communication between elements of the external function stations, main memory, I/O channels, arithmetic unit, registers and console. A translator accepts an emulated instruction and then determines the address in control

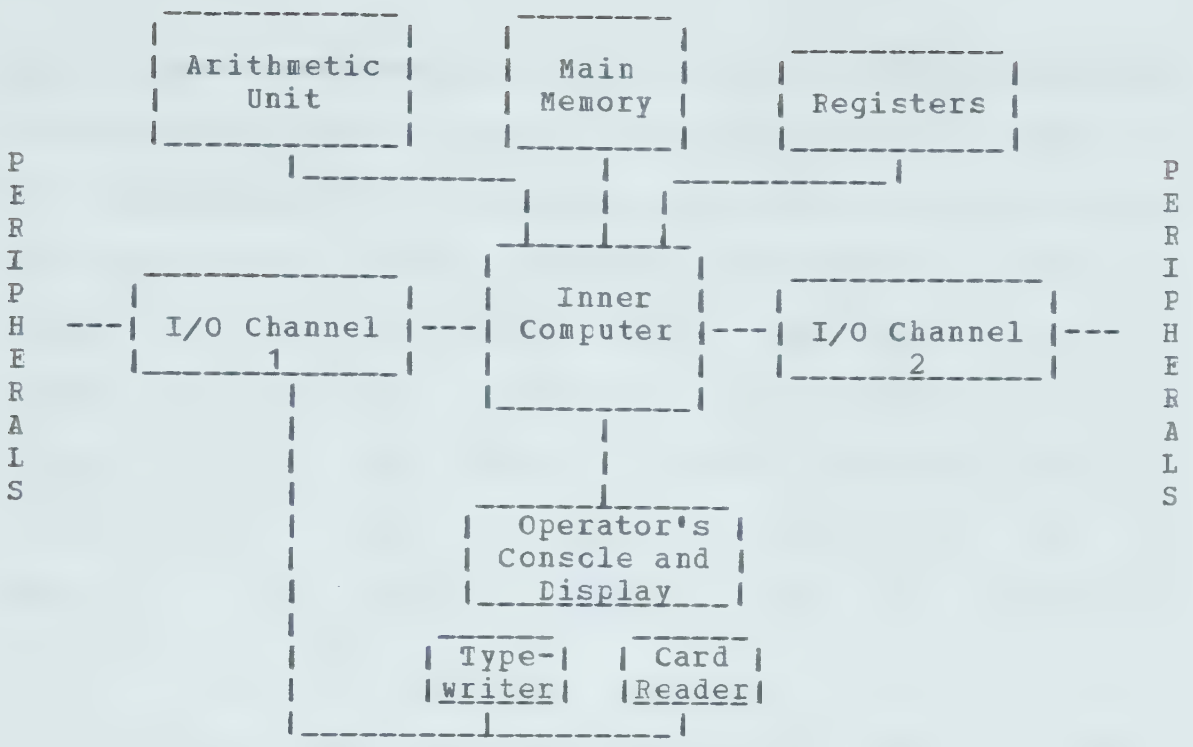

Figure 2.2: MLP-900 General Configuration

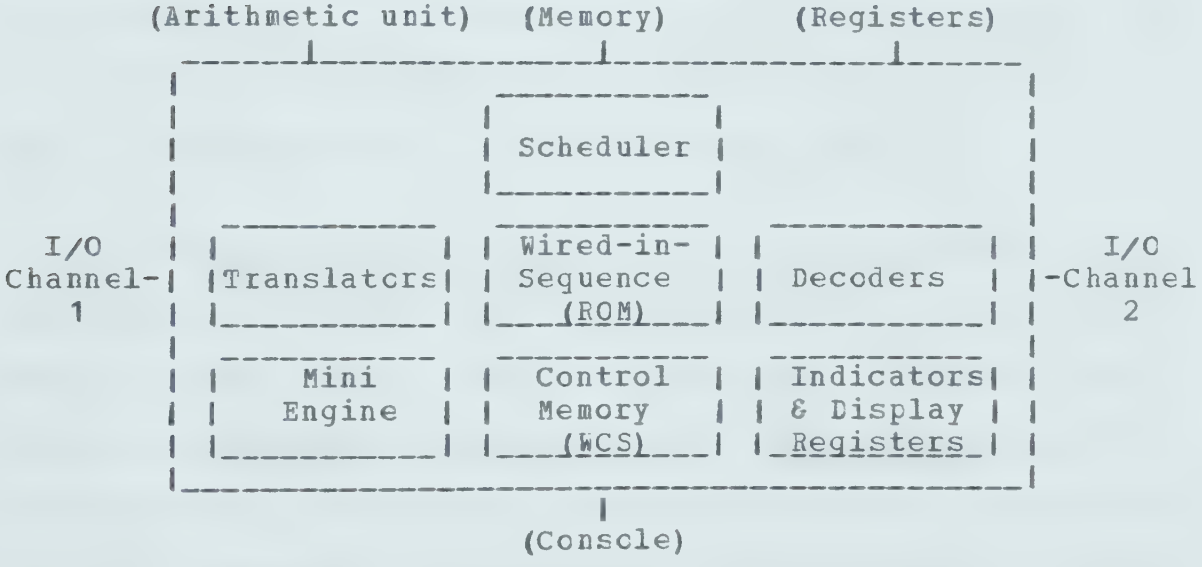

Figure 2.3: MLP-900 Inner Computer Configuration

memory where the microroutine begins. The encoded fields in the microinstruction are handled by the decoders to generate the remainder of the line pulses. The mini engine controls the action of the wired-in-sequence (ROM segment of control memory) and writable control memory. The MLP-900 like the system 360 has quite extensive microdiagnostics, which provide a very high degree of fault resolution with the minimum amount of operating hardware (refered to as the hardcore of the machine) necessary for the diagnostic routines to function.

The most recent addition to the field of WCS is IBM's system 370/165 [13], which employs monolithic writable control store in addition to ROM. The ROM contains most of the instruction set and some specialized routines, while the WCS contains the remainder of the instruction set, the series 7000 emulators, and the diagnostic routines.

Microdiagnostics on the 370/165 include resident and nonresident routines. The resident routines provide error checking of the input lines to the more extensive non-resident diagnostic routines, which are kept on magnetic disk cartridge and read by a dedicated I/O device when the CPU is in diagnostic mode. The use of bootstrapping further minimizes hardcore requirements.

In addition to CPU microprogramming, system 370 uses

microprogramming for its 3830 disk controller. The microroutine loading procedure is similar to that of the CPU.

The two systems, MLP-900 and 370/165, are state of the art implementations of WCS. These two are the largest systems currently employing microprogramming to this degree. Greater development in state of the art implementations can be seen in minicomputers. For example, Microsystems have introduced a completely microprogrammable machine, the Micro -1600, while Nanodata Corporation released the specifications for their microprogrammable machine QM-1, early in 1971.

The QM-1, developed as a microprogramming laboratory machine, is described by Rosin et al [25]. Nanodata in the QM-1 provide a two level structure, a nanostore and a microstore. Their nanoprograms, stored in the nanostore, are horizontal, polyphase nano-primitives which are employed to write routines for the microstore. The use of a two level control store allows architectural integrity as desired by the manufacturer while offering the flexibility of the microprogrammed system to the user. The nano-primitives are the critical design problem and must provide for a wide spectrum of control signals so that flexibility is not sacrificed for design convenience or hardware optimality. The definition and implementation of user nano-

primitives is possible on the QM-1 system.

Presently microprogramming enjoys a very large audience in the computer industry and specifically the user populace. Microprogramming offers, for certain applications, a viable alternative to both hardware and software.

2.3 Microprogramming vs Hardware

Clapp in his article Applications of Microprogramming [5] enumerates four categories of comparison between microprogramming and hardware: flexibility, cost, speed and reliability.

The greatest advantage microprogramming has over hardware is the flexibility inherent in the host machine, allowing future unforseen requirements to be accommodated with a minimum of hard machine modification. Existing capabilities can be upgraded to make use of new algorithms or procedures. Hardware procurements can be standardized for the multiple computer user; i.e. a single hardware machine can be purchased and tailored to meet his diverse needs. By the use of emulators, a host machine can be microprogrammed to represent the architecture of any machine.

A microprogrammed computer enjoys several cost advantages over conventional CPUs. A standardized hardware design leads to savings in the design and fabrication times. Waldecker [33] in comparing a conventional and a microprogrammed CPU found that logic requirements, excluding control memory, were within 5-10 per cent of each other with 85 per cent of the microprogrammed machines logic suitable for large scale integration(LSI), as opposed to 50 per cent of the conventional CPU. Further cost savings can be expected in microprogrammed CPU's through forecast cost reductions in LSI fabrication(Hornbuckle [10]). Lower hardware maintenance costs are expected for microprogrammed machines, because of reduced technician training time and high resolution microdiagnostic routines. Duplication of hardware facilities can be reduced by the assignment of many tasks to the microprogrammed machine. The replacement of high quality, relatively inexpensive maintenance free equipment by a computer and operator to prevent duplication can be uneconomical if an unequitable substitution is made.

In general, speed is the hardware CPU's strongest advantage whereas in a microprogrammed CPU to achieve equivalent speeds one or a combination of the following techniques are used:- parallel, polyphase, or encoded control. Increasing the amount of parallel execution increases the complexity of the data paths and cost.

Polyphase execution gives a speed increase but also increases the ROM word width which increases cost. Encoding, which reduces the ROM word width, gives a speed increase but requires more complex and less flexible decoding hardware. In a well designed hardware machine the CPU can be expected to operate at memory speeds, whereas the speed of a microprogrammed CPU is dependent on the number of microinstructions executed in one main memory cycle and their relative power.

The orderly structure of a microprogrammed computer provides a reliable structure that can be fabricated with fewer unique hardware components, allowing a simpler construction procedure to be used. In multiple computer applications, a common hardware base for each component of the system can reduce the amount of hardware backup required. A single microprogrammed computer can be used to back up, with the appropriate microprograms, all computers in a series of applications.

The flexibility of a microprogrammed computer leads directly to cost and reliability advantages over hardware. Although the execution speed of a microprogrammed sequence is slower than an equivalent hardware implementation, the development of user oriented microprogrammed operation codes can improve application performance. This will be the case if the developed codes are used in a high proportion to

other codes. For some applications a cost disadvantage can be encountered with the use of microprogrammed control. For example, in a timesharing environment where interruptability is necessary, very complex microroutines could lead to poor resource utilization for the whole system. IBM on the 370 series for their new instruction, move character long (MVCL, moves up to 16 Megabytes at once) have provided for the interruptability of this instruction so that the CPU can continue to service other interrupts.

2.4 Microprogramming vs Software

Clapp [5] has done a very good analysis of this area and has compared microprogramming and software on the following points: speed, cost and reliability.

He enumerated the attributes of software functions which if present will generally yield a speed increase when microprogrammed:

1. CPU bound functions, not dependent on input/output transfer time.
2. Functions with intermediate results which do not have to be preserved after the function is complete.
3. High usage sequences.
4. A microinstruction that provides an easy way to do an operation that is difficult with the

regular machine's instruction set(falls beyond the power of the current instructions but is not logically prohibited).

5. Sequences which are high in address generation functions and low in execution time.

A speed increase can almost always be expected if a software routine is micrcccded. Patzer and Vandling [18] have done a comparison of microprogrammed and software square root routines. The microroutine demcnstrated a constant cost per bit of the result, whereas the software routine had a decreasing cost per bit as the number of bits increased. The absolute cost per bit remained larger for the micrcroutine.

The cost of a microprogrammed computer is dependent on the extent of application in the system. Two goals are possible:- complement, or replace the software system. Microprogramming tc ccmplement the system can lead to cost savings in the form cf:- faster software development, easier debugging prccedures and mcre efficient source software. To program all cr a large part of an operating system in microcode will be far more expensive, less rigorously tested and very difficult to modify [5]. Firmware will be more expensive tc prcduce than equivalent software, until a standardized high level microprogramming language is developed.

Low hardccre requirements and the high degree of system integrity offered by firmware makes it far more reliable than software systems. Read/write protection can be offered with firmware routines at considerable savings over comparable software protection. Special instructions used by multiprogramming environments can be made uninterruptable with firmware implementation.

2.5 Summary

Microprogramming has evolved in computer systems due to the interest of the manufacturers in the cost advantage offered by micrcprogrammed machines. The advantages of microprogrammming such as flexibility, cost and speed(software) outweigh the disadvantages, the lack of portability and compatability(upwards). How and where microprogramming is used in the future is dependent upon the reaction of the manufacturers and the users to the increasing viability of microprcgramming as an alternative.

CHAPTER III

APPLICATIONS

3.1 Introduction

The advantages of microprogramming, as outlined in Chapter II, do not in themselves predefine what will be a successful application. Manufacturer and user have diverse requirements while trying to achieve the same objective, a cost effective system. The user sees microprogramming in the role of a software support facility and as such would like the capability of accessing and modifying the microcode. At present the manufacturer sees microprogramming as an exclusive tool for the implementation of functions which he determines to be of value. This is based on observation of the techniques used to microprogram and the quantity of information made availible to the user. The areas of application that the manufacturer and user groups have currently found successful will now be examined. Even though they do benefit both sides, some of the areas have been divided and placed where this author feels is most appropriate. Section 3.4 looks at the use of

microprogramming in the design of computer systems. Section 3.5 deals with a cross examination of the major areas of application and their implications.

3.2 Manufacturers Applications

The manufacturer of hardware computer systems has generally employed microprogramming to maximize the cost benefit to himself. The areas where microprogramming have been used most beneficially are; direct operating system support, microprocessors, microdiagnostics and emulators. These areas are oriented toward benefitting the manufacturer more than the user.

3.2.1 Operating System Support

Modern operating systems, multiprogramming and time sharing, require many of their routines to be executed repeatedly. Clapp [5] has said that instructions with high execution frequency can be microprogrammed to good advantage. Werkheiser [34] has set out three criteria for microprogramming a process in an operating system:- have a high frequency of use, must be well defined, and be beyond the scope of the virtual machine instructions. For their operating systems the manufacturers have microprogrammed some of their system routines. In general the easier routines have been implemented. These include bootstrap

loaders, store and fetch protection, interrupt traps, subroutine linkage and instruction re-try. The lack of general operating system procedures being implemented can be attributed to the high cost, in time and programming effort, to develop the microroutines.

Huberman in The Principles of Operation of the Venus Microprogram describes the microprograms and software routines used to implement a multiprogramming operating system. The philosophy of that development group was that the work required for the implementation can be greatly reduced by having an architecturally compatible machine. Their operating system supports named virtual memories, recursive and reentrant routines, debugging interrupts and a microprogrammed multiplexor channel. The instruction set is completely defined by microprograms. Microroutines were written to perform job management functions:- CPU scheduling, job queue management and virtual address to real address mapping. When a page fault is found, the memory mapping microprogram calls a software routine to handle the page-in/page-out function. The results of this project were that by allowing the architectural specifications to influence the design, the effort in software development was reduced substantially.

Replacement of operating systems with microprogrammed routines negates the past development and man-hours

dedicated to produce these software systems.

3.2.2 Microprocessors

The small general purpose CPU, referred to as a microprocessor, is microprogrammed for low cost and high flexibility. They are suitable for coupled CPU networks where processor hierarchies exist.

Hornbuckle and Ancona [10] describe the LX-1 (ECM) computer which is microprogrammable. The LX-1 is small and flexible and can be used as a peripheral controller, special purpose machine or as described in the article, a real time signal processor.

The Microprogrammed Control Unit as described by Roberts et al [22] is a high speed, executing input output processor and interrupt handler for the Naval Research Laboratory Signal Processing Element(SPE). The design criteria was speed but not at the expense of flexibility. The control unit was required to be able to control all hardware elements in the SPE. A microprogrammed approach was chosen as it met the requirement for flexibility and speed [22].

IBM [13] uses a microprocessor with WCS in system 370 for their disk controller. Peripheral controllers are ideally suited to microprocessor application. IBM's most

recent announcement(1973) included the 3704 communications controller as a replacement for the 270x series. The 3704 can run in emulation mode to emulate the 270x series or in the Network Control Program(NCP) mode. It is a stored program line controller(max 64K local memory). Integrated I/O employs additional circuitry within the processor under microprogram control to replace present high cost channels. Maholick and Schwarzell [16] examine this type of control.

3.2.3 Microdiagnostics

Part of the firmware support routines include diagnostic checks on the performance of the hardware logic. These routines cannot check the logical consistency of the system's architecture or detect the presence of intermittent and transient faults. Logic failures are the prime target of microdiagnostics.

The primary reasons for current microdiagnostic usage are to provide controlled abnormal termination, dynamic system reconfiguration in case of faults, and a high degree of fault resolution. Writable control store usage has greatly increased the spectrum of the microdiagnostic routines that a system can support [19]. Widespread use of microdiagnostics will take systems one step closer to being fault tolerant and self repairing [19].

3.2.4 Emulators

In the design of emulators, solution of four major problems is the key to success;

1. Mapping the image of the target machine into the host machine. Defining the target machine in terms of the host machine's resources.
2. Selection and implementation of machine instructions to be microprogrammed, to complement the host machine's instruction set and facilitate the target machine definition.
3. Linkage from emulation mode to host mode of operation. Most systems require a control store load(physically or electronically) and an initial program load(IPL, the initial stages of operating system start up) with the target machine's system parameters.
4. Develop an operating structure under which the emulator is run.

In addition to general overall design problems, I/O emulation has several special problems;

1. One to one mapping of the subject I/O devices into the I/O system. The physical and logical definitions of the target I/O devices in terms of the physical host devices.
2. Data conversion and translation from the

target I/O routines to the host I/O devices.

3. Linkage between CPU and I/O(interrupts or time outs).
4. Overlapping I/O and CPU emulation for better target machine performance.
5. Exceptional conditions. Problems that must be solved with additional hardware or software.

Integrated emulators as introduced by IBM [13] solve the major problems of I/C emulation and emulator design. These emulators run as problem programs under the control of the operating system and offer several advantages leading to better resource utilization in a multiprogramming environment through;

1. Reduction in the number of IPLs, no mode switching for emulators. An IPL although requiring only two minutes to perform does cause the system to be stopped. Meaning before the IPL, all executing jobs must terminate and no new jobs can be accepted before the IPL.
2. Greater system flexibility, online systems can run continuously without interruption by emulation mode processing.
3. Better input-output handling. I/O modules for each emulator accept the I/O calls then interpret and issue calls to operating system access methods.

Machine mapping and operation code selection are the major problems of emulator design. I/O emulation is now reduced to the creation of the interface routines between the system I/O packages and the emulator.

3.2.5 Conclusions

The use of microprogramming in operating systems, microprocessors, emulators and microdiagnostics have long useful lives compared to development time, therefore optimization (hard microprogramming) of the control signals is economic.

3.3 User Applications

The user views microprogramming as a cost effective means whereby he can create a machine which will more directly meet his needs. At the time of purchase a computer system can be an unknown quantity. Power and versatility are qualities that the salesman has attached to a particular machine without necessarily explaining their implications. As familiarity with a particular machine grows, the user begins to appreciate the possible applications that can be tried and where his emphasis should be put. Only then does he realize the real meaning of power and versatility in that

system. The concept of power and flexibility will be examined in more detail in a later chapter. However, his machine must have the power to perform his tasks economically and the versatility to perform a broad spectrum of these tasks. What should a user look for in a computer architecture so as to guarantee his future financial and technological position? Does his present system provide features that will be supported in the future?

3.3.1 Operation Code Compatibility

Operation code compatibility over a single manufacturer's line of computers is a characteristic of third generation computer systems. IBM offers it with microprogrammed machines, while Univac and CDC offer slower hardware for the lower end of their lines.

Compatibility benefits the user for two reasons:- minimizes program conversion and personnel retraining costs. For these reasons, the user is able to stay with state of the art equipment in a developing line of computers, and can thus achieve better performance at a lower cost(shown in the decreasing costs for computer hardware, e.g. on the IBM 370 series, monolithic memory is half of the price for core memory).

Rosen [25] and Sammet [28] predict for the future a

greater polarization toward problem oriented languages. Operating system designers are turning to high level system programming languages, such as PI360 [37], Algol on the Burroughs computers, and BLISS [38] for the larger DEC machines.

Why then is it important to have operation code compatability? The reason is efficiency. Using Dijkstra's [6] concept of levels of abstraction in computer systems, programming languages can be considered levels of abstraction of the real machine. The more abstract or further from the machine, the less efficient is the code. Microcode or hardware control signals are closest, machine code next, assembly languages, high level and problem oriented languages last. Direct language oriented machines circumvent the problem of abstraction but are less flexible since only problems suited to expression in the language of the machine are readily solved.

Operating systems must be efficient for reasons of cost. System overhead or the amount of processing performed by the operating system relative to problem processing time must be kept to a minimum. From an internal point of view large systems require many man-years development, so in order to defray these costs over a reasonable period of years, the future systems must support an upward compatible operation code set.

3.3.2 Special Purpose Machines

Every time a user loads a program into a general purpose machine he views it as a special purpose machine to solve his particular problem. He wants something that will execute his task efficiently through sound design, rather than the use of extravagant computer resources. A suggested implementation is WCS with dynamic mode switching for efficient resource utilization. In this way a single processor can be an efficient economical solution to the need for a special purpose machine.

3.3.3 Speed and Flexibility

The user can look forward to speed increases with microprogrammed machines if a multi-moded approach is used. Microprogrammed control is inherently slower than hardware control therefore any speed increases have to be achieved by the advantageous use of the microprogrammed machine's flexibility. The user can achieve cost gains(reduced source coding time, smaller design to production lag time on a well designed application machine) through the application of microprogramming to his specific problem.

3.3.4 Conclusions

The areas of direct application and concern of the user can be classed as soft microprogramming implementations. A particular user application will in general have a very short life(based on the parallel development of conventional software programs which have an expected life of 30 months [14]) and thus requires a short lead time to be economically viable. As a result, for microprogramming to be an alternative to software the instructions must be of a vertical nature(easy to use and versatile).

3.4 Computer Design

Bell [3] points to microprogramming as a desirable tool for the design of logic systems, which is consistent with the original concept as advanced by Wilkes [36]. The advantage of microprogramming in the design of logic based systems is the regularity which is incorporated into the design. This is further supported by the suitablity of microprogrammed machines to LSI fabrication techniques [33].

Stabler [31] shows that it is possible to have, by transformations on the states of the control unit, machines that are all operations unit (hardware machine) or have all signals generated by the control unit (totally microprogrammed). He suggests that the use of these

transformations without computational assistance will be impossible. The formal definition of the operations unit, control unit and the transformations will make machine assistance possible. Schlaeppi [29] describes the development of the LOTIS language for logic, timing and sequencing description. He proposes that this language is suitable for hardware system descriptions and the synthesis of dataflows. The work in this field shows a trend to automatic computer design.

3.5 Summary

The solution to both manufacturers and users application problems is to provide two levels of control memory. ROM for hard microprograms and WCS for the soft applications of the user. While the solution may be obvious, its very limited use would point to a reluctance on the part of the manufacturer to allow the user access to microcode. Up to the present time Standard Computer Corporation (MIP-900) had the only large system with this type of microprogramming. IBM on their system 370/165 do have this hardware configuration but the use of WCS is limited to the manufacturer. Smaller systems, Micro Systems 1600, Nanodata's QM-1 and Burroughs 1700 provide user access.

Application areas of interest that have been examined

previously in this chapter will now be considered for their effects on the other party.

The use of microprogramming by the manufacturer for direct operating system support will be quite transparent to the user. The main benefit to be gained by its application will be a reduction in system overhead. System throughput should increase, since a greater amount of CPU time is available for the execution of the users tasks. Large systems will make available support routines in RCM, while in the smaller systems, these routines will be designed and written by the user, if he wants them.

The increasing demand for highly dependable economic computer systems (Amdahl [2]), will necessitate the use of high resolution diagnostics in large systems. The ability to resolve a fault quickly and precisely will be the measure of a machine's repairability. Microdiagnostics provide a desirable level of fault resolution at a reasonable price. The low hard core requirements and systematic approach to fault isolation due to bootstrapping make microdiagnostics an attractive choice for diagnostic implementation. The inclusion of microdiagnostics in large systems has already been adopted in the IBM system 370 models 155-165. In small systems the microdiagnostics might be a ROM board that a technician brings for his maintenance checks. It can be expected that the inclusion of microdiagnostics in a small

system will be an optional extra (at additional cost).

Large system manufacturers will provide IBM type integrated emulators, which are more efficient in computer resources and easier to use than traditional emulators. They will be provided on a pre-installment basis allowing present work to be moved to the new system without emulation. Personnel training and familiarization for the new system can be conducted in-house. With adherence to the principle of operation code compatibility by the major manufacturers, the need for emulators from generation to generation could vanish. The use of emulators will allow a user who has considerable investment in equipment of a particular manufacturer, to use computer products manufactured by another company. Emulation will reduce conversion costs and allow an easier transition. The emulation of hardware peripherals will become more involved with virtual systems and new access methods(IBM - 3705 emulates the 270x controller series, in non-emulate mode it uses virtual telecommunications access method whereas 270x uses the basic telecommunications access method).

Operation code compatibility has become a goal of computer system design and precludes the in-house development of special operation codes by the user. Microinstructions, unlike the high level and assembly languages, will not generally be upwards compatible because

of their dependence on technology. Sequences of micro-operations which previously could be executed in parallel, in a later system due to timing might require redevelopment to be executable on the new system. The advent of a high level microprogramming language will help circumvent this problem. Therefore special instructions developed at present by the user will have to be reprogrammed when a change in systems is made.

Manufacturer produced special purpose machines using microprogramming will lead to a more economical application than a user produced machine. Short term special requirements are more suited to software solutions. Special purpose machines are not as dependent upon the flexibility of the new machine as a general purpose system. The trade off considered is the cost of the flexible general purpose machine against the performance actually required.

The use of microprogramming facilities will be constrained by the additional programming effort required to produce microcode, both programmer training and actual coding time and the lack of a guarantee by the manufacturer to support any nonstandard software packages. This makes the user with in-house developed microcode not only tied to a particular manufacturer but quite possibly a particular machine. For the preceding reasons it would seem reasonable that microprogramming facilities on computer systems will be

left in the hands of the manufacturers. Minicomputers are the possible exception to the manufacturer's control, and educational institutes can benefit from extensive microprogramming facilities, in the study of computer architecures, telecommunications hardware, system performance monitors and language processors.

CHAPTER IV

FUTURE DEVELOPMENTS

4.1 Introduction

The use of microprogramming in computer structures will be examined in this chapter. A discussion will be included of the power versus flexibility trade off and how microprogramming produces a workable solution.

System structures that are presently going through a development stage are cache memory, and pipelined, multiprocessor and array computers. The implications of possible inclusicn of microprogramming in these will be considered. The well developed structures; mini computers, telecommunication equipment and time sharing services are examined for growth and how microprogramming can facilitate their implementation. System dependent features, system performance mcnitors and programming languages are discussed for their implementation by microprogramming.

4.2 Power vs Flexibility

The concept of power in modern general purpose computers is poorly defined. Computing power is at present the measure of the number of fetch-decode-execute cycles a processor can perform in a unit of time. It includes no measure of instruction set richness or operating system performance. While not a direct measure of computing power the richness of the instruction set is an indicator of the computer's ability to solve problems in the various areas of application (flexibility). Similarily, the operating system performance gives an indication of the proportion of the computer's power available for the solution of the user's jobs (efficiency).

The objective of a general purpose computer system is to solve problems efficiently for a broad spectrum of applications. Economic considerations involved in the design and construction of a CPU dictate that the instruction set must be limited, at the cost of the machine's flexibility. A CPU dominant computer gives an efficient solution to CPU bound application areas, while a flexible balanced computer configuration gives more uniform performance characteristics over a wider spectrum of applications. The balancing of CPU resources to the external resources, main memory access time, channel activity, number of channels will yield a more flexible and

economic total computing resource.

The addition of WCS to a flexible computer allows the implementation of special operation codes to give performance peaks in major application areas of the user. An area of general interest where special operation codes are useful (as mentioned in Chapter III) is the operating system itself.

4.3 Developing Architectures

The factors considered in the architectures discussed are cost, flexibility and reliability and how microprogramming can help. The more specific areas such as solution to implementation problems are beyond the scope of this thesis.

4.3.1. Cache Memory

A cache memory is a small high speed memory used in computer systems to reduce memory access time. The executing program keeps some of its data and instructions paged into the cache memory from where the processor executes the program. On a data exception (data not in cache), data is requested from the slower primary memory. Bell [3] says that; "in cache-based computers ... simple conventional instructions and microinstructions are

essentially identical and execute at the same rate." He goes on to point out that; "... caches can also be used to hold user microprograms."

A conventional instruction can be considered a highly encoded, polyphase, serial (parallel, depending upon instruction preprocessing capability of the CPU) microinstruction. Since the instructions are highly encoded, a cache structured processor is far less flexible than an equivalent WCS processor (flexibility is lost as the amount of encoding increases, see Chapter I.). Therefore the only advantage of a cache memory over a WCS is its high speed write capability which is not always necessary for control memories.

Conceptually then a cache memory is not in fact a unique building block for future architectures but a special application of WCS. A case in point is Nanodata's QM-1 computer which has a high speed microstore (WCS) that can be obtained in quantities great enough so that the WCS can be used as a cache. The control store is availible in 4K blocks up to 32K.

4.3.2 Pipelined Computers

Pipelining is the employment of sequential job stations to perform one or many functions on a vector of operands.

Station one performs step one of an instruction then passes its result to station two for continued processing, etc. Station one itself is now able to accept another input and continue processing. A pipelined structure allows for high usage of the individual job stations (much like an assembly line) and a greater total system throughput, provided the number of functions executed of that type are high.

Microprogrammed control for this structure is possible, but at the expense of the performance of an individual sequence. However, the cost would go down and the resulting structure would have potentially greater flexibility and reliability. The hardware under control of a microprogram could be shared with other operations. To be cost-effective the most expensive component of the system must be the one with the greatest activity (40 per cent of the cost for the 370/195 CPU was for the pipeline multiplier, Foster [8]). A floating point multiply is performed in an average instruction mix 3.8 per cent of the time [8], with a multiply to add ratio of approximatly 10. Therefore an upper bound of about 28 per cent of CPU active time is spent in the multiply sequence, thereby suggesting the use of a pipelined multiplier in an average environment would be excessive. This bound is found by assuming the other 96.2 per cent of the instructions are adds, therefore 134.2 adds are executed in the average mix of which 38 are multiply

equivalents. The use of microprogramming for this structure will depend on a cost benefit analysis which considers the hardware costs, desired performance ratio, and the application of the computer. A special purpose high performance computer will degrade the importance of cost whereas the use of pipelined structures in general purpose computers will require more weight be given to cost in the analysis.

4.3.3 Multiprocessors

A multiprocessor environment has many processors sharing resources (data bases, peripherals). Structurally this architecture can be very reliable due to the redundancy in processors. Unlike past computers the major cost component of modern systems is not the CPU, but rather the cost of peripherals, main memory and telecommunications equipment.

The use of microprogramming to implement the processors for this environment can produce gains in diagnostic procedures, reduction of cost/processor at the expense of the individual processor's speed. The speed cannot be considered critical (Bell [3] shows that with the addition of four processors the speed increases by a factor of three). Therefore a slight loss in speed/processor can still be expected to yield an overall increase in

performance.

4.3.4 Array Structured Processors

Array structured processors have many identical processing elements (PE's) all governed by the same controller, executing the same function in parallel on different data. Provided the problem is suited to solution by this type of system great speed increases can be expected, but for normal problems the speed is no better than conventional systems. If excessive preprocessing is performed to reformulate an unsuitable problem losses in efficiency can be expected [3].

Microprogrammed PE's each with their own control store for microinstruction storage and intermediate results will serve two purposes; to provide a local store, and to make the respecification of processing element functions easier and therefore the processing elements themselves more flexible. The controller itself, while best suited to hardware implementation due to speed considerations, could be microprogrammed.

4.4 Current Areas

General systems already developed in hardware or software are now examined, especially where the application

of microprogramming will, cost or performance wise, enhance the cost/perfcrmance ratio. As these systems evolve, the requirements cf greater flexibility and reliability are becoming dominant. Therefore microprogramming merits consideration as an alternative to current implementation techniques.

4.4.1 Telecommunications

The use of stored program message switching centers for telecommunications is increasing. The advantages for this type of switching being;

1. Flexibility - the ability to add, modify or replace switching functions after original design and implementation without necessarily making hardware modifications.
2. Tailorability - can be made to meet the exact needs of the user at the time of installation.
3. Maintainability - software routines can monitor calls for problems, perform diagnostic checks for errors, and detect faulty circuits, reroute calls and continue processing with the equipment in error flagged and removed from operation.

In the area of data communications, monitoring for path retry, alternate path selection, line error analysis and

summary will remove a load that is currently held by a CPU. Maintenance can be scheduled by line error summary statistics therefore employing a preventative approach to errors rather than an after the fact repairing approach.

Software controlled switching, even though it allows the ability to modify, add or replace functions, means that the machine architecture is specified at the time of fabrication, so certain alternatives for future modifications are excluded. Software maintenance routines allow the logical performance of the switching center to be checked, but not the functioning of the processor. Exhaustive software diagnostic routines would require long execution time, large core requirements and a high hardcore requirement. Microprogramming the CPU would provide the fault resolution (microdiagnostics) and the flexibility to add options that previously are not available with a software-hardware system.

Additional areas where microprogrammming could be used to good effect are:-

1. Intelligent terminals - low cost local processing terminals which can perform editing functions, line checking and code translation.
2. Data capture facilities - remote terminals for input to larger systems on a network. These terminals can be data input(warehouse

inventory input) or real time data collection (pipeline monitoring signals).

3. Remote facilities maintenance - remote equipment with the use of microdiagnostics could be more reliable with diagnosis being initiated by the terminal or the central controller.

The field of telecommunications promises to be an area where microprogramming can be used to good advantage. The capability of a microprogrammed terminal can be superior to an equivalent cost hardware teminal.

4.4.2 Minicomputers

As previously stated the application of microprogramming technology in minicomputers is high. The cost advantages in the small systems provide to the user a good cost/performance ratio. The cost savings are translated into performance increases in the form of more comprehensive instruction sets, integrated communication control (to replace channels), float-point multiplication microroutines, or special user dependent application routines. A microprogrammed minicomputer can be tailored to meet the application needs of the user. There is no reason to believe that widespread use in minicomputers will not continue.

4.4.3. Time Sharing Services

A growing trend towards large, shared, remote data bases (accessed through centralized time sharing services) is focusing attention on design and implementation considerations for system reliability and ease of use. The solution to the reliability problem is suggested to be parallel operation, two CPU's performing the same job in a linked fashion, one being the primary system with the second running as a background system to inherit the primary tasks in case of failure. Cost effectively this type of system is inefficient. The reliability inherent in microprogrammed systems has been previously discussed (Chapter II) and provides a high level of reliability at a cost effective price.

4.5 System Dependent Usage

The usage discussed in this section deals with areas of application that are system dependent. The actual implementation can vary from system to system so that the independent advantages of microprogramming in system performance monitors and programming languages are given. Operating system support is system dependent, but because of its thorough examination in Chapter III will not be included.

4.5.1 System Performance Monitors

The increased attention being given to hardware system reliability in the form of hardware, software and firmware diagnostics has been in parallel with a growing emphasis on operating system and software efficiency. Performance monitoring consists of two distinct procedures, data collection and data analysis (usually at different times). The state of the art for the implementation of system performance monitors is hardware and software [15]. Shively [30] mentions the possible use of microcode as a means of data collection. The flexibility required in the analysis of the data necessitates the use of software.

To reflect the true activity of an operating system, a monitor must run without biasing the data it is collecting. Hardware monitors give completely unbiased data while software introduces an estimable level of bias at best. Data collection with software has the flexibility desired for the complete spectrum of a system activity whereas hardware, due to cost, must be of narrower scope.

Microprogramming has greater flexibility than hardware and can be used in conjunction with additional registers and a buffer to run concurrently with the executing microroutine, thus eliminating the bias of software collection. In the worst case the bias of a microprogrammed

monitor can be measured rather than estimated as with the software monitor. The buffer can be cleared by normal system I/O or extracted by an external processor for later processing. This area promises to be one of greater interest for microprogramming application.

4.5.2 Language Processors

Language interpretation can be facilitated easily and efficiently on a microprogrammed processor by coding the source instructions of the language as microroutines. The return to interpretation as a method for language processing has been advocated by Rosin [26]. Interpreters are suitable to an environment where the number of different jobs is high, the execution time of each job is low, the turn around time is short and the majority of programs are one shot programming efforts (typical multi-user time sharing facility). Usage in this area has proven to be successful, for example SAIT (student assembler language translator) at the University of Alberta, [7].

Microprogramming for the implementation of an interpreter is more flexible than hardware and faster than software. Problem oriented languages can be structured in a macro form so that a macro-processor can be implemented in control memory to facilitate the execution of the program. A multiprogramming environment can be preserved by the use

of integrated language processors, the same technique used by IBM with their emulators.

Although microprogramming is most directly applicable to interpretation, compilers can make use of microprogrammed routines to speed the compilation process (arithmetic expression parser, table handling).

4.6 Summary

Above and beyond all the points made in this chapter, microprogramming can be used in the design of all logic based systems.

The area of telecommunications would seem to be one of the most important in the field of computing science. The cost, reliability and flexibility advantages offered point to an increasing amount of microprogramming use. Future system architectures, even though exotic, can be microprogrammed to provide advantages to small systems. This can be considered a test for system architectures. If the advantages remain as such when implemented as a small system then the architecture is significant. Whereas if they disappear, the sophisticated hardware implementation was the dominant element of the architecture.

CHAPTER V

CONCLUSIONS

The preceding chapters have looked at the reasons for increased interest in microprogramming. How, why, where and when microprogramming grew to prominence has been discussed leaving only one question left, who? By whom, I mean what groups can use microprogramming and how? Whom do the disadvantages of microprogramming affect the greatest and conversely who do they affect the least? The obvious groups to examine are those in Chapter III, manufacturer and user. Users are classified as small, medium and large according to their computer size and activity. The manufacturer's usage is broken into two classifications, Research and Development, and production. The production aspects of the manufacturer's usage has been examined thoroughly in Chapter III. An additional group has been added, the Educational Institutes (more directly Departments of Computing Science). They exhibit the same characteristics as a medium to large user and the research section of the manufacturer.

Microprogramming presents three disadvantages to the

potential user.

1. At present microprograms are not generally portable from one computer to another. For instance, a set of programs written for a particular hardware machine will in general only be portable to a different physical machine that is a hardware equivalent.
2. Microprograms written for one machine of a computer line will not be upwards or downwards compatible.
3. The development time for microprograms will vary depending upon the amount of parallel and horizontal programming in the code, but will be greater than for a comparable software system [5].

These disadvantages point to a guideline for the usage of microprogramming. In the absence of all other objectives and constraints, microprogramming should be used for long term applications to provide functions that will be available on a future upgraded facility. Building a sophisticated system around a microprogrammed function (most probably at greater expense) which may or may not be available in a future system could be a uneconomical decision. The future of WCS has not fundamentally been decided and could ultimately become a dedicated tool of the manufacturer (indirect advantages accruing to the user; such

as microdiagnostics and extended operation codes).

The previously defined categories, user and manufacturer, will now be examined for objectives and criteria which might override the above guideline.

A small system user (minicomputer, minimum of peripherals) has one or two major systems that use the majority of his running time. He typically supports one high level language and an assembler. Modern technological trends have brought the CPU costs down until the biggest per cent of his capital expenditure is on peripherals which in turn are his limited resource. Integrated I/O communication provides the best area for usage. Should he decide to link to another system, microprogram controlled communications could provide better response and data preparation (decoding, deblocking). The need for a WCS microprogramming facility would be a special case for the small system user, such as replacement of several distinct elements with similar hardware but architecturally different units. A microprogram controlled CPU (ROM) would be the only way to approach this group's needs.

The medium system user (medium range processor, 360/40/50, DOS, Tapes) supports a greater number of systems, runs a greater number of hours/week and in general makes greater use of his system than the small user. He, like the

small user, only supports one or two high level languages (Cobol, Fortran or PL1) and an assembler. He is a very cost conscious user and is always willing to consider ways to upgrade his performance without hardware purchases. His desire for a cost-effective system makes him naturally attracted to a microprogrammed facility. Microprogramming's offer of greater system life, more throughput with minimal hardware purchasing (cost for microprogram development extra) and a high degree of system reliability look very attractive. A major area of concern for the medium sized user is his internal growth pattern. If his computer usage should grow rapidily, he does not want to have extensive microprogrammed support. Whereas if he should be running close to his load limit and feels he will be running at this level for the foreseable future, then microprogrammed support and its benefits could be considered. In considering microprogrammed support his fundamental tradeoff is his future compatibility versus increased efficiency. Without compatibility he can be certain that he will not be able to run his programs on another system in case his system fails. For the medium sized user to make extensive use of a WCS microprogramming facility he would have to have a special problem (functions that are real time critical, economic reasons to extend the capability of his machine). ROM (including microdiagnostics) with faster hardware (hardware multiplier etc.) seem to satisfy this group's needs.

The large system user (370/155-165, OS-MTS, large data bases, RJE, online data capture) runs a great variety of systems, supports the majority of the high level languages and is very security conscious. The integrity and reliability of the large system are the two most important criteria when the selection of a system is being attempted. WCS and the ability for dynamic modification for the operation code set can lead to a high degree of resource consumption for protection mechanisms. Reliability is enhanced with the inclusion of IBM 370/165 type microdiagnostic routines which are overlayable into the WCS. The large user will use WCS and ROM to provide basic elements of the system he wants whereas the availability of WCS will be restricted.

Manufacturer usage of microprogramming for the production of computer hardware and systems has been discussed in Chapter III. The results pointed to the following areas where the usage can feasibly be made:- operating system support, microdiagnostics, emulators and microprocessors. In the area of Research and Development the manufacturer can employ microprogramming to complement his current procedures for:- algorithm testing and debugging, operation code design and study, analysis and emulation of paging structures and page replacement

algorithms. The application of microprogramming in the laboratory for this research can lead to shorter design lead time and a faster system development cycle.

The Educational Institute was mentioned in the summary of Chapter III as a possible exception to any control that manufacturers might decide to apply to the use of WCS. The reason for this is obvious, research. What are the areas that a University with a CM-1 like machine will be able to investigate?

1. Computer architecture design and evaluation - evaluate new architectures and develop unique designs of their own (without the extensive backup now required for such investigations e.g. Illiac IV).
2. Operation code set evaluation - what new operation codes can be added to create a universal set and why these particular ones.
3. System performance monitors - monitor and evaluate operating systems. The design and study of monitors.
4. Telecommunications - study and evaluate communication networks, intelligent terminals, RJE terminals and message switching.
5. Operating system design and evaluation - paging algorithms, internal queue management strategies.

These avenues for research are vital for the growth of a Computing Science Research Institute. The use of microprogramming allows for the study and evaluation of the hardware domain of computer systems. An algorithm like an architecture should maintain its characteristics independent of environment, so software simulation can suffice. Simulation while feasible is not adequate for study in these areas. Microprogramming and emulation satisfy the desire for hardware test facilities without the attendant costs.

The two major users of microprogramming (WCS) in the future will be the manufacturer and the research oriented institute. The future points to microprogramming as an economic tool to provide flexibility, reliability and performance.

REFERENCES

1. Allen, M. W.; Pearcy, T.; Penny, J. P.; Rose, G. A.; and Sanderson, J. G. "CIRRUS, An Economical Multiprogram Computer with Microprogram Control." IEEE Transactions on Computers, Vol. C-12, No. 12 (December, 1963), 663-671.

2. Amdahl, L. "Architectural Questions of the Seventies." Datamation, Vol. 16, No. 1 (January, 1970), 66-68.

3. Bell, G. C.; Chen, R.; and Rege, S. "Effect of Technology on Near Term Computer Structures." Computer, Vol. 5, No. 2 (March/April, 1972), 29-38.

4. Benjamin, R. I. "The Spectra 70/45 Emulator for the RCA 301." Communications of the ACM, Vol. 8, No. 12 (December, 1965), 748-752.

5. Clapp, J. A. "The Application of Microprogramming Technology." Sigmicro Newsletter, Vol. 3, No. 1 (April, 1972), 8-47.

6. Dijkstra, E. M. "The Structure of the 'THE' Multiprogramming System." Communications of the ACM, Vol. 2, No. 5 (May, 1968), 341-346.

7. Dutton, J. B. Student Assembler Language Translator: System Description, Computing Center Publication, University of Alberta, 1969.

8. Foster, C. "A View of Computer Architecture." Communications of the ACM, Vol. 15, No. 7 (July, 1972), 557-565.

9. Glushkov, V. M. "Automatic Theory and Formal Microprogramming Translations." Cybernetics, Vol. 1 (September, 1968), 1-8.

10. Hornbuckle, G. D., and Ancona, E. J. "The IX-1 Microprocessor and Its application to Real Time Signal Processing." IEEE Transactions on Computers, Vol. C-19 (August, 1970), 710.

11. Huberman, B. J. "Principles of Operation of the Venus Microprgram." The MITRE Corporation, MTR 1843, ESD-TR-70-198, Contract F19(628)-68-C-0365, Bedford, Mass., (May, 1970).

12. Husson, S. S. Microprogramming: Principles and Practices. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1970.

13. IBM Systems "A Guide to the IBM System/370 Model 165." Document No. GC20-1730-0.

14. Lichstein, H. A. "When Should You Emulate?" Datamation, Vol. 15, No. 11 (November, 1969), 205-207.

15. Lucas, H. C. "Performance Evaluation and Monitoring." Computing Surveys, Vol. 3, No. 3 (September, 1971), 79-92.

16. Maholick, A. W., and Schwarzell, H. H. "Integrated Microprogrammed Communication Control." Computer Design, Vol. 8, No. 11 (November, 1969), 127-133.

17. Opler, A. "Fourth Generation Software." Datamation, Vol. 13, No. 1 (1967), 22.

18. Patzer, W. J., and Vandling, G. C. "Systems Implications of Microprogramming." Computer Design, Vol. 8, No. 12 (December, 1967), 62-67.

19. Ramamoorthy, P.; Chang, D. "Modeling and Testing for Microdiagnosticc." IEEE Transactions cn Computers, Vol. C-21 (november, 1972), 1169-1183.

20. Rakowczi, L. L. "The Computer-Within-A-Computer, A Fourth Generation Concept." IEEE Computer Group News, Vol. 3, No. 2 (1969), 14-20.

21. Redfield, S. R. "A Study in Microprogram Processors: A Medium Sized Microprogram Processor." IEEE Transactions on Computers, Vol. C-20, No. 7 (July, 1971), 743-750.

22. Roberts, J. D.; Ihnat, J.; and Smith, W. R. "Microprogrammed Control Unit(MCU). Programming Reference Manual." Sigmicro Newsletter, Vol. 3, No. 3 (October, 1972),18-58.

23. Rosen, S. "Programming Systems and Languages: A Historical Review." Programming Systems and Languages. Edited by Saul Rosen. New York: McGraw-Hill,Inc., 1967.

24. Rosen, S. "Electronic Computers: A Historical Survey." Computing Surveys, Vol.1, No. 1 (March, 1969), 7-36.

25. Rosen, S. "Programming Systems and Languages 1965-1975." Communications of the ACM, Vol. 15, No. 7 (July, 1972), 591-600.

26. Rosin, R. F. "Contemporary Concepts of Microprogramming and Emulation." Computing Surveys, Vol. 1, No.4 (December, 1969), 197-212.

27. Rosin, R. F.; Frieder, G.; and Eckhouse, R. H., Jr. "An Environment for Research in Microprogramming and Emulation." Communications of the ACM, Vol. 15, No. 8 (August, 1972), 748-760.

28. Sammet, J. E. "Programming Languages: History and Future." Communicatcns of the ACM, Vol. 15, No. 7 (July, 1972), 600-610.

29. Schlaeppi, H. "A Formal Language for Describing Machine Logic, Timing and Sequencing." IEEE Transactions cn Computers, Vol. EC-13 (August, 1964), 439-448.

30. Shively, R. R. "Performance Evaluation." Computer, Vol. 5, No. 5 (September/October, 1972), 12-15.

31. Stabler, E. "Microprogram Transformations." IEEE Transactions on Computers, Vol. C-19 (October, 1970), 908.

32. Tucker, S. G. "Microprogram Control for System/360." IBM System Journal, 6 (1967), 222.

33. Waldecker, D. E. "Comparison of a Micro-programmed and a Non-microprogrammed Computer." Computer Design, Vol. 9, No. 6 (June, 1970), 73-78.

34. Werkhieser, A. H. Microprogramming the Operating System. Preprints 3rd Workshop on Microprogramming, Buffalo, New York, October, 1970.

35. Wilkes, M. V. The Best Way to Design an Automatic Calculating Machine. Report of Manchester University Computer Inaugural Conference, July, 1951.

36. Wilkes, M. V. " The Growth of Interest in Microprogramming: A Literature Survey." Computing Surveys, Vol. 1, No. 3 (September, 1969), 139-145.

37. Wirth, N. "PL360, A Programming Language for the 360 Computers." Journal of the ACM, January 1968,37-74.

38. Wulf, W. A. et Al., BLISS Reference Manual, Department of Computer Science, Carnegie Mellon University, Pittsburgh, 1970.